DATA 8005 Advanced Natural Language Processing

Efficient LM Adaptation

DATA 8005 Advanced Natural Language Processing

LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

[Yatai Ji]

Fall 2024

Background

- Pre-training → Downstream Adaptation
- Model size: larger and larger
- Convenience: shared pre-trained language model for multiple applications

- Low-Rank Adaptation (LoRA): inject trainable rank decomposition matrices into each layer of the Transformer architecture

Existing methods

- Full fine-tuning: low-efficiency
- Adapter: introduce Inference Latency

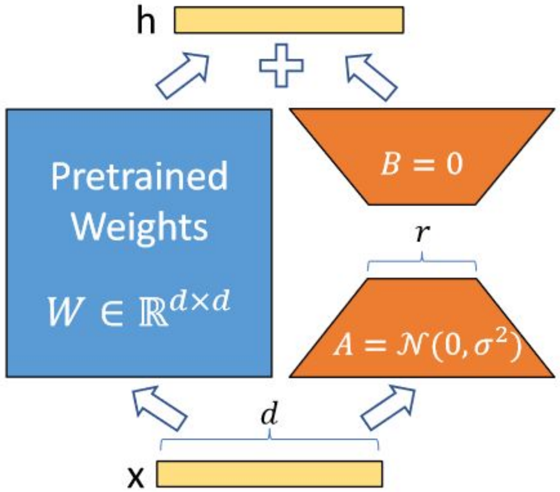
Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
Adapter^{L}	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^{H}	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

- Prefix-tuning: hard optimization, reduces the sequence length available to process a downstream task
- posing a trade-off between efficiency and model quality

Motivations

- The learned over-parametrized models in fact reside on a low intrinsic dimension.
- The updated weights have a low "intrinsic rank" during adaptation.
- Constrain the ranks of updated weights.

$$W_0 + \Delta W = W_0 + BA$$



LoRA

- Training some dense layers indirectly by optimizing rank decomposition matrices of the dense layers' change
- Tuning: W0→W, update △W == freeze W0, learn △W (BA, to constrain low-rank)

$$h = W_0x + \Delta W x = W_0x + BAx$$

- A: random Gaussian initialization; B: zero, so ΔW = BA = 0
- Increase r, training LoRA roughly converges to full tuning.
- Merge: explicitly compute and store W = W0 + BA, No Additional Inference Latency
- Only adapting the attention weights and freeze the MLP modules

Advantages

- A pre-trained model can be shared and used to build many small LoRA modules for different tasks.
- LoRA makes training more efficient and lowers the hardware barrier. GPT-3 175B fine-tuned, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times.
- No additional inference latency: merge the trainable matrices with the frozen weights when deployed.
- LoRA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning.

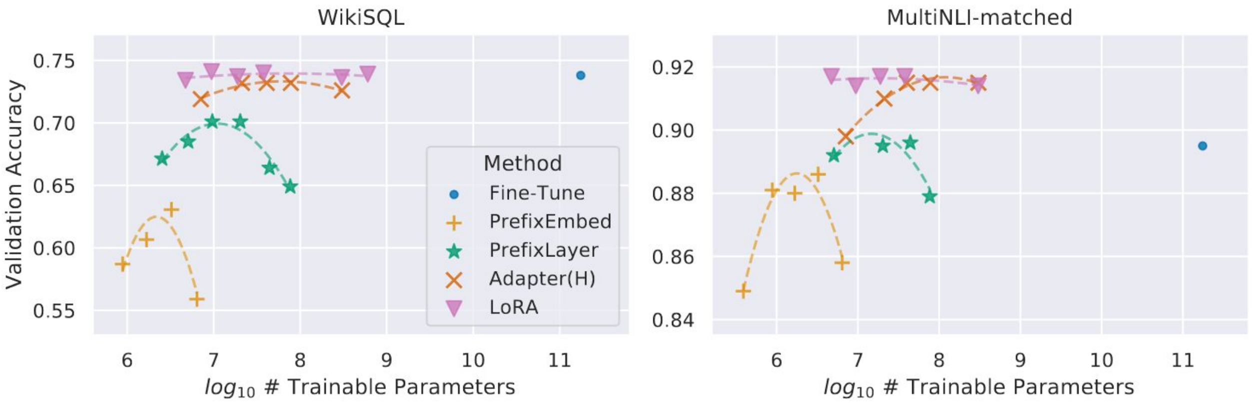
Experiments

- Comparison: Fine-Tuning (FT), Bias-only or BitFit, Prefix-embedding tuning (PreEmbed), Prefix-layer tuning (PreLayer), Adapter tuning, LoRA

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} ($Adpt^{D}$)*	0.3M	87.1$_{\pm.0}$	94.2$_{\pm.1}$	88.5$_{\pm1.1}$	60.8$_{\pm.4}$	93.1$_{\pm.1}$	90.2$_{\pm.0}$	71.5$_{\pm2.7}$	89.7$_{\pm.3}$	84.4
RoB_{base} ($Adpt^{D}$)*	0.9M	87.3$_{\pm.1}$	94.7$_{\pm.3}$	88.4$_{\pm.1}$	62.6$_{\pm.9}$	93.0$_{\pm.2}$	90.6$_{\pm.0}$	75.9$_{\pm2.2}$	90.3$_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	87.5$_{\pm.3}$	**95.1**$_{\pm.2}$	89.7$_{\pm.7}$	63.4$_{\pm1.2}$	**93.3**$_{\pm.3}$	90.8$_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
RoB_{large} (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB_{large} (LoRA)	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	91.6$_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
RoB_{large} ($Adpt^{P}$)†	3.0M	90.2$_{\pm.3}$	96.1$_{\pm.3}$	90.2$_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	83.8$_{\pm2.9}$	92.1$_{\pm.7}$	88.4
RoB_{large} ($Adpt^{P}$)†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	89.7$_{\pm1.2}$	67.8$_{\pm2.5}$	**94.8**$_{\pm.3}$	91.7$_{\pm.2}$	80.1$_{\pm2.9}$	91.9$_{\pm.4}$	87.9
RoB_{large} ($Adpt^{H}$)†	6.0M	89.9$_{\pm.5}$	96.2$_{\pm.3}$	88.7$_{\pm2.9}$	66.5$_{\pm4.4}$	94.7$_{\pm.2}$	92.1$_{\pm.1}$	83.4$_{\pm1.1}$	91.0$_{\pm1.7}$	87.8
RoB_{large} ($Adpt^{H}$)†	0.8M	90.3$_{\pm.3}$	96.3$_{\pm.5}$	87.7$_{\pm1.7}$	66.3$_{\pm2.0}$	94.7$_{\pm.2}$	91.5$_{\pm.1}$	72.9$_{\pm2.9}$	91.5$_{\pm.5}$	86.4
RoB_{large} (LoRA)†	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.2**$_{\pm1.0}$	68.2$_{\pm1.9}$	**94.8**$_{\pm.3}$	91.6$_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	**91.9**$_{\pm.2}$	96.9$_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

Experiments

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1



More Understanding

- Which weights

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

- Optimal rank r

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

- LoRA potentially amplifies the important features for specific downstream tasks that were learned but not emphasized in the general pre-training model.

Discussions

- Adaptively adjust rank r?
- Model compression with rank decomposition?

DATA 8005 Advanced Natural Language Processing

QLoRA : Efficient Finetuning of Quantized LLMs

Sidi Yang

Fall 2024

Motivation

- Finetuning LLM is effective!
- Regular 16-bit finetuning of a LLaMA 65B parameter model requires more than 780 GB of GPU memory
- But with QLoRA, we can do this in a single 48GB GPU!

Background

- What is quantization

90

100

85

Bucket： A

Bucket： B

- Quantization: Converting FP32 to INT8

FP32: - ∞ ~ + ∞

$$\text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right)$$

Int8： -127~128

NormalFloat4

- Something different



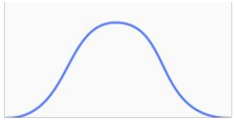
100　　100　　99

Bucket：A

NOBODY

Bucket：B

- Considering the distribution

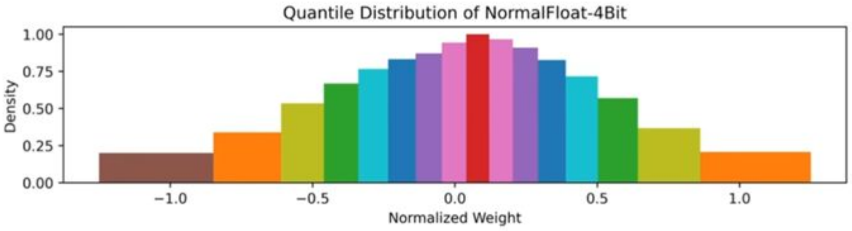




FP32: - ∞ ~+ ∞

NF4：16 buckets

NormalFloat4

- Asymmetry: negative and positive



- Quantization constant: Scalar
- Lookup tables

[1.532, -0.823, 0.412, 0.232]

scaling

[1. , -0.537, 0.269, 0.151]

Lookup tables

[15. , 2 , 10 , 9]

Quantization

Lookup tables

[1. ,-0.525, 0.246, 0.160]

scaling

[1.532 ,-0.804, 0.377, 0.247]

De-Quantization

-1.0	0
-0.6961928009986877	1
-0.5250730514526367	2
-0.39491748809814453	3
-0.28444138169288635	4
-0.18477343022823334	5
-0.09105003625154495	6
0.0	7
0.07958029955625534	8
0.16093020141124725	9
0.24611230194568634	10
0.33791524171829224	11
0.44070982933044434	12
0.5626170039176941	13
0.7229568362236023	14
1.0	15

Double Quantization

- Blocksize for 4-bit quantization: 64

- For storing the scalar: 32-bit

- 32/64 = 0.5 extra bits for each parameter

- 8-bit Quantization for the scalar $$8/64 + 32/(64 \cdot 256) = 0.127 \text{ bits}$$

- Reducing 3GB for 65B model.

QLoRA Fintuning

- Gradients of LoRA is needed

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}}, \quad (5)$$

- Contains the gradient of weight

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \quad (6)$$

- The weight of NF4 is dequantized to BF16 for caluculation

Paged Optimizer

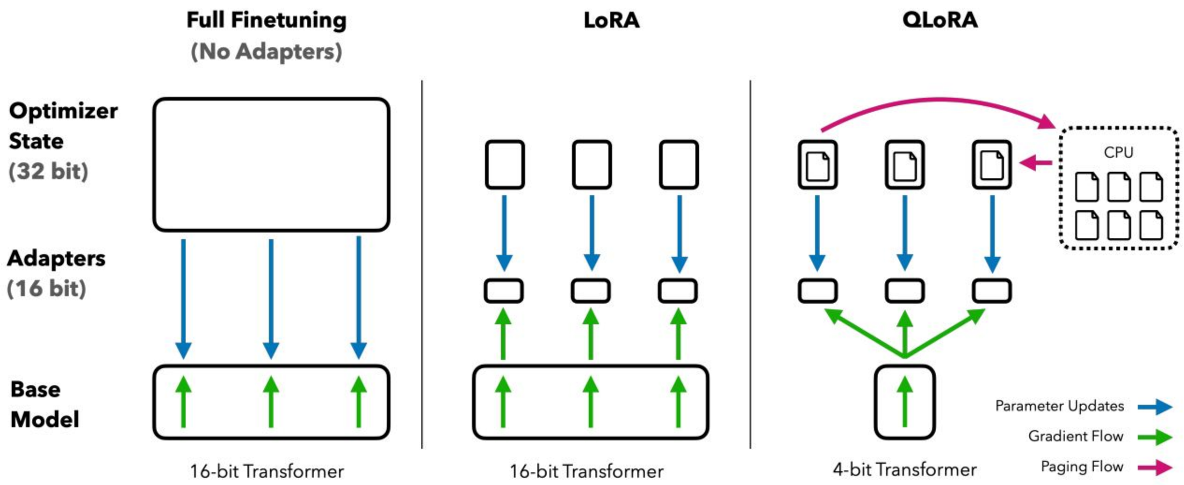
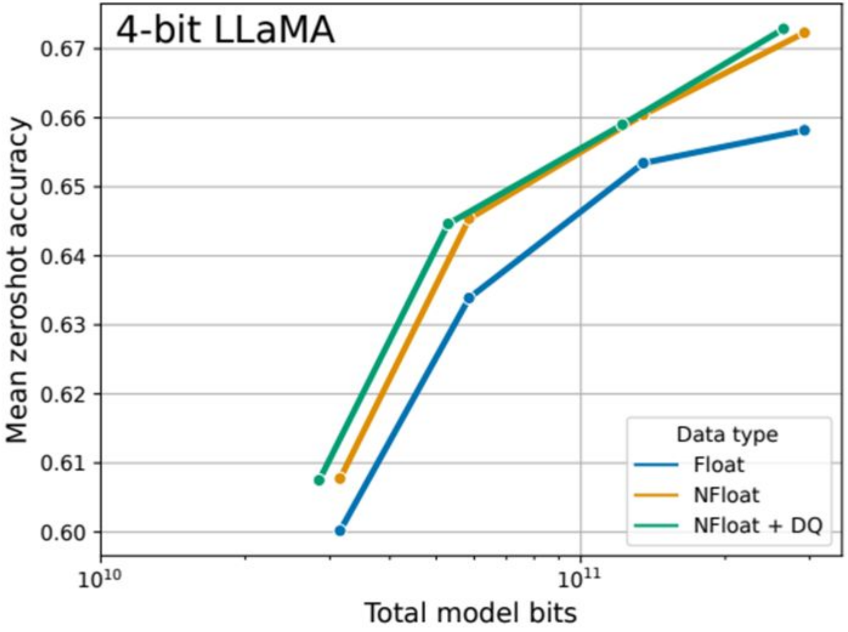


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

- Preventing the out-of-memory of GPU

Experiment

- Comparison: NF4 v.s. FP4



Experiment

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

- Imprecise quantization can be fully recovered through adapter finetuning after quantization

Evaluation

- Automated Evaluation GPT-4 and human evaluation
- ELO: the expected win-rate relative to an opponent's win rate
- A tournament-style evaluation

Model	Claude 3 Opus	GPT-4
Context size	200,000	8192
Cutoff date	Aug 2023	Dec 2023
Input/Output cost	$15/$75	$30/$60
Max output tokens	4096	4096
Latency (TTFT)	1.99s	0.59s
Throughput	25 t/s	125 t/s

Experiment

Table 6: Zero-shot Vicuna benchmark scores as a percentage of the score obtained by ChatGPT evaluated by GPT-4. We see that OASST1 models perform close to ChatGPT despite being trained on a very small dataset and having a fraction of the memory requirement of baseline models.

Model / Dataset	Params	Model bits	Memory	ChatGPT vs Sys	Sys vs ChatGPT	Mean	95% CI
GPT-4	-	-	-	119.4%	110.1%	**114.5**%	2.6%
Bard	-	-	-	93.2%	96.4%	94.8%	4.1%
Guanaco	65B	4-bit	41 GB	96.7%	101.9%	**99.3**%	4.4%
Alpaca	65B	4-bit	41 GB	63.0%	77.9%	70.7%	4.3%
FLAN v2	65B	4-bit	41 GB	37.0%	59.6%	48.4%	4.6%
Guanaco	33B	4-bit	21 GB	96.5%	99.2%	**97.8**%	4.4%
Open Assistant	33B	16-bit	66 GB	91.2%	98.7%	94.9%	4.5%
Alpaca	33B	4-bit	21 GB	67.2%	79.7%	73.6%	4.2%
FLAN v2	33B	4-bit	21 GB	26.3%	49.7%	38.0%	3.9%
Vicuna	13B	16-bit	26 GB	91.2%	98.7%	**94.9**%	4.5%
Guanaco	13B	4-bit	10 GB	87.3%	93.4%	90.4%	5.2%
Alpaca	13B	4-bit	10 GB	63.8%	76.7%	69.4%	4.2%
HH-RLHF	13B	4-bit	10 GB	55.5%	69.1%	62.5%	4.7%
Unnatural Instr.	13B	4-bit	10 GB	50.6%	69.8%	60.5%	4.2%
Chip2	13B	4-bit	10 GB	49.2%	69.3%	59.5%	4.7%
Longform	13B	4-bit	10 GB	44.9%	62.0%	53.6%	5.2%
Self-Instruct	13B	4-bit	10 GB	38.0%	60.5%	49.1%	4.6%
FLAN v2	13B	4-bit	10 GB	32.4%	61.2%	47.0%	3.6%
Guanaco	7B	4-bit	5 GB	84.1%	89.8%	**87.0**%	5.4%
Alpaca	7B	4-bit	5 GB	57.3%	71.2%	64.4%	5.0%
FLAN v2	7B	4-bit	5 GB	33.3%	56.1%	44.8%	4.0%

Discussion

- Did not establish that QLORA can match full 16-bit finetuning performance at 33B and 6D5B scales

- How about 3-bits or other quantization?

DATA 8005 Advanced Natural Language Processing

LoRA Learns Less and Forgets Less

Jing Xiong

Fall 2024

Introduction

Research Background

- Memory and Compute Demands

  LoRA, Prefix-Tuning, etc.

- Training Time and Efficiency

  Converge with fewer epochs or samples.

- Risk of Forgetting

  Trade of performance on target domain tasks and catastrophic forgetting during training.

Challenges in Low-rank Fine-tuning

- Risk of Forgetting

Fine-tuning in domain-specific areas often leads to substantial forgetting, but this issue remains unclear in the domain of complex reasoning.

- The trade-off between learning and forgetting

It remains unclear what the trade-off is between low-rank fine-tuning methods and full parameter fine-tuning methods.

- How does its pattern compare to full-parameter fine-tuning on complex reasoning tasks?

How does its performance compare to full-parameter fine-tuning on complex reasoning tasks?.

Introduction to Low-Rank Adaptation

- Low-rank approximation of the fine-tuning perturbation matrix

$$W_{\text{finetuned}} = W_{\text{pretrained}} + \Delta$$
$$\Delta = \gamma_r AB, \quad A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times k}.$$

- Targeted Module

$$W_q^{(l)}, W_k^{(l)}, W_v^{(l)}, W_o^{(l)}$$

- How is catastrophic forgetting different in LoRA?
- How does LoRA perform in complex reasoning settings?

Motivation

- How is catastrophic forgetting different in LoRA?
- How does LoRA perform in complex reasoning settings compared to full-parameter fine-tuning?
- How does LoRA perform in complex reasoning settings and general conversational training scenarios?
- What is the trade-off between the degree of forgetting in the current model and its performance in the target domain?

Experiment

Setup

- Model

Llama-2-7B

- Datasets

Code（HumanEval, StarCoder-Python, Magicoder-Evol

Math（OpenWebMath, MetaMathQA, GSM8K）

- Setting

Continued pretraining (CPT)

Instruction finetuning (IFT)

Full finetuning is more accurate and sample-efficient than LoRA

- In CPT, LoRA underperforms full finetuning across all configurations

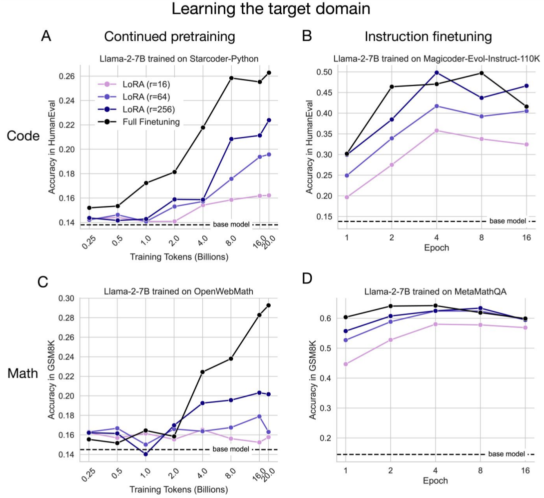
  Full fine-tuning consistently outperforms LoRA in performance.

  As the number of tokens for tuning increases, the performance gap continues to widen.

- In IFT, high LoRA ranks are required to close the gap with full finetuning

  High ranks are required to close the gap with SFT, especially in code (rank=256).

  LoRA is sample-efficient in code datasets but not in math, requiring more training epochs for comparable performance.



LoRA forgets less than full finetuning

- Metric

  HellaSwag: Describe an event with multiple possible continuations.

  WinoGrande: Assesses commonsense reasoning.

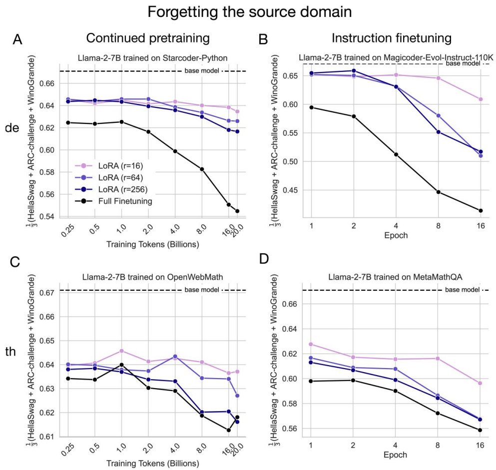
  ARC-Challenge: Tests complex reasoning and understanding of scientific concepts

- IFT induces more forgetting than CPT.
- The extent of forgetting is controlled by rank.

  In code – for both CPT and IFT – full finetuning forgets substantially more than any LoRA configuration.

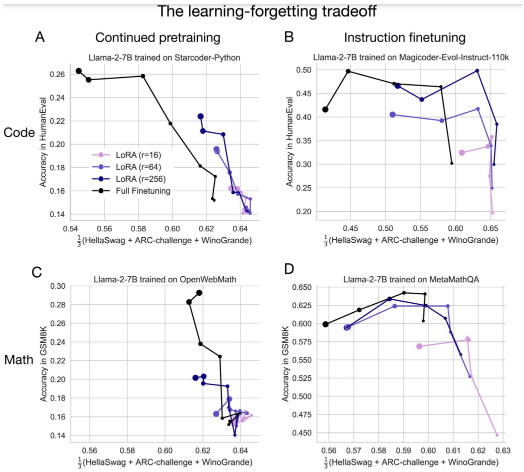
  In math – for both CPT and IFT – LoRA with r = 256 forgets nearly as much as full finetuning.

  Lower ranks forget less.



The Learning-Forgetting Tradeoff

- Each dataset presents a unique tradeoff pattern



- IFT induces more forgetting than CPT.

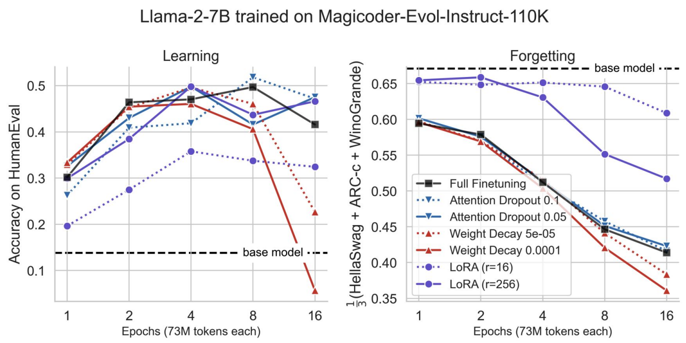
For Code CPT, LoRA and full fine-tuning perform similarly but with more forgetting in fine-tuning, while for math CPT, both overlap until full fine-tuning later achieves higher GSM8K scores without forgetting

For code IFT, LoRA (r=256) matches full fine-tuning in accuracy with less forgetting, while in math IFT, full fine-tuning provides a better learning-forgetting balance.

Fine-tuning results on the general SFT dataset

- Chat quality is similar to full parameter fine-tuning performance.

  Multi-Turn Benchmark, GSM8K, Massive Multitask Language Understanding.



- LoRA exhibits less forgetting.

Other interesting observations

- LoRA forgets less than attention dropout and weight decay.

- LoRA helps maintain diversity of token generations.

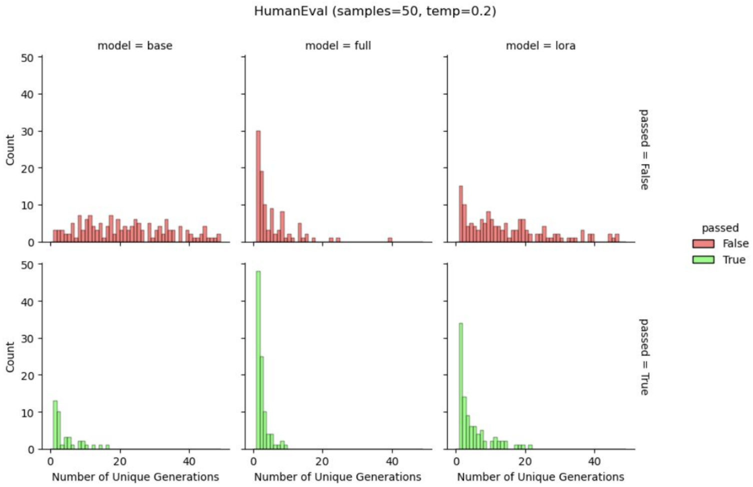
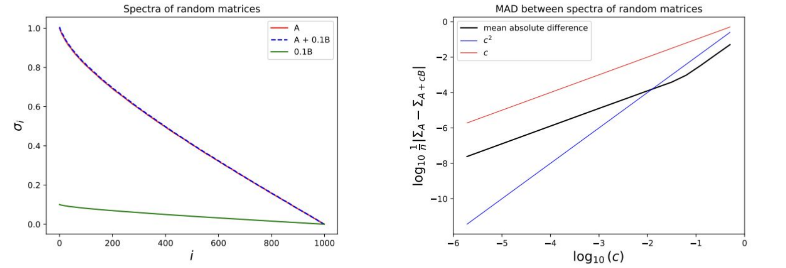


Figure 5: **LoRA maintains output token diversity relative to full finetuning.**



(a) Spectrum of A and $A + cB$ as well as cB for $c = 0.1$. Notably, $A, cB, A + cB$ are all high rank.

(b) Mean absolute difference between spectra of A and $A + cB$ for various c.

Figure S8: **Analyzing the spectra of the sum of two** 1000×1000 **Gaussian i.i.d matrices**. A and B are 1000×1000 random matrices with i.i.d. standard normal Gaussian entries.

Perturbations Matrix observations

- Critically, the difference Δ has a similar spectrum to the finetuned and base weight matrices (up to a multiplicative scaling).

- There is nothing extraordinary about the full finetuning spectra; similar spectra can be achieved by adding low-magnitude Gaussian i.i.d noise to a weight matrix.

- The rank of the matrix continues to increase as fine-tuning progresses.

Discussions

- Can the rank of the perturbation matrix truly reflect the information increment after fine-tuning?
- Can we obtain a Pareto optimal solution between model forgetting and target domain performance by measuring the information of the perturbation matrix and the original matrix?
- Why do the phenomena of forgetting and learning differ so significantly between the code and data domains?